Infusion of Autonomy Technology into Space Missions: DS1 Lessons Learned

Abdullah S. Aljabri[1], Douglas E. Bernard], Daniel L. Dvorak', Guy K. Man[1], Barney Pell[2], Thomas W. Starbird'

[1] Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
818-354-5862
abdullah.s.aljabri@jpl.nasa.gov

[2] RIACS
NASA Ames Research Center
Moffett Field, CA 94035
650-604-3361
pell@ptolemy.arc.nasa.gov

Abstract—The impact of infusing breakthrough autonomy technology into a flight project was a big surprise. Valuable technical and cultural lessons, many of general applicability when introducing system-level autonomy, have been learned by infusing the Remote Agent (RA) into NASA's Deep Space 1(DS1) spacecraft. The RA's architecture embodies system-level autonomy in three major components: Planning and Scheduling, Execution, and Fault Diagnosis and Reconfiguration. Lessons learned include: The architecture was confirmed. Active participation by non-autonomy personnel in the development is essential. Communication of new concepts is essential, difficult, and hampered by differences in terminology. Giving a spacecraft system-level autonomy changes organizational roles in operating the spacecraft after launch, and hence changes roles during development. Software models supporting functions traditionally handled on the ground must be developed early enough to get on-board. Shortfalls in planned features must be technically and developmentally accomodatable, in particular not to threaten the launch schedule. Traditional commanding must be supported. Testing must be emphasized; end-to-end tests counter skepticism. These lessons and others, on incremental system releases and use of autocode generation, are based on 16 months of spiral development from start of project through the project's decision to reduce the role of the RA from full-time control of the spacecraft to a separable experiment.

TABLE OF CONTENTS

1. INTRODUCTION

The NASA New Millennium Program (NMP) is designed to flight-test new break-through technologies that have a high impact on space science in the 2 1st century. The Program sets very high goals and the participating teams are challenged to go outside of their comfort *zone,* This paper reports on such an example: the development for flight validation of the Autonomy Remote Agent (RA) technology. The RA is a joint effort between JPL and NASA Ames. The autonomy team set high goals for itself, consistent with the broad, system-level nature of the application, and with the tight schedule of the Deep Space 1 (DS 1) flight project. There have been major accomplishments, and use of the technology has proceeded on DS 1, though not as originally planned. The nature of the challenges faced were multifaceted and they are all important to technology infusion. One key area we underestimated was the cultural challenge in using a breakthrough technology.

1 Historically, major advances in spacecraft autonomy were done within a flight project and introduced by the spacecraft and mission operations team. The advancements are evolutionary in nature and are mostly defined by mission needs. In DS 1, although the Remote Agent extends spacecraft autonomy based on previous accomplishments, it does so in a big step with major involvement from technologists. The autonomy enabled by the Remote Agent in DS1 is system-level, not confined to a single subsystem such as autonomous pointing by the Attitude Control Subsystem. The Remote Agent transforms the entire spacecraft from a traditional open loop system as seen from the ground to a "closed-loop" system, and there are many ramifications in this approach.

The purpose of this paper is to document major lessons learned from the Herculean endeavor to achieve a significant technology advance under the New Millennium Program. On the one hand, the Remote Agent team failed to recognize the extent of the *systems and cultural impact* of doing something radical. This underestimation of impact was one contributing cause for the DS1 project to cease to rely on the Remote Agent as the controller of the spacecraft for the entire mission, reducing its role to a week-long experiment. On the other hand, the Remote Agent team also has validated a major part of the approach and technologies on the ground, significantly advancing the technology. The DS 1 lessons are

invaluable for the Remote Agent team and the autonomy effort in NASA for ongoing and future work and these lessons will also be valuable to any major attempt at technology infusion where *marry people are affected.* The lessons were gathered from the Remote Agent team consisting of technologists, spacecraft engineers, mission engineers and managers. While many lessons were learned by other teams, in this paper, we chose to focus on only those issues within the control of the team. There were many other choices within the control of other teams which also contributed to the situation.

2. The New Millennium Program

NMP focuses on new technologies that contribute significantly to reducing the cost while increasing the relative scientific capability of future space and Earth science missions. Its key areas of focus are to lower mass to reduce launch cost; to lower life-cycle costs to increase mission frequency; and to reduce operations costs through greater spacecraftautonomy.

The NMP will enable 21st century science missions through the identification, development, and flight-validation of key advanced technologies. Breakthrough technologies selected from the existing technology pipeline—made up of technology programs of NASA, other government agencies, industry, nonprofit organizations, and academia-will be developed in partnership with these organizations. These critical technologies will be validated so that future science missions can take advantage of them without assuming the risks inherent in their first use. NMP technology-validation flights will also provide opportunities to capture meaningful science. New technologies will be infused into the nation's commercial base, and significant benefits to US industrial competitiveness will be realized through the joint development program. The New Millennium Program will also pioneer new ways of partnering with industry, nonprofit organizations, and academic institutions.

There are several NMP validation flights under development. On DS 1 is the first deep space flight in a trajectory representative of a realistic science mission, the DS 1 spacecraft will fly by asteroid McAuliffe in January, 1999 and comet West-Kohouteck-Ikemura in June, 2000. A flyby of Mars is also planned between the asteroid and the comet encounters. The DS 1 spacecraft, scheduled to be launched in July, 1998, is jointly developed by NASA and the industry partner Spectrum Astro, Inc. At project start, October 19, 1995, DS1 was to carry and validate thirteen new technologies, including: Autonomy Remote Agent, Miniature Integrated Camera and Spectrometer, onboard Optical Navigation, 3-D Stack Flight Computer, Solar Electric Propulsion and Solar Concentrator Arrays. In March, 1997, the planned role of the Autonomy Remote Agent was changed, and the 3-D Stack Flight Computer was deleted from DS 1 (see Section 6.). There are three autonomy technologies on DS1 and this paper covers the development lessons for the Autonomy Remote Agent.

3. DS 1 Autonomy/Flight Software Architecture

A software architecture diagram [1] of the DS 1 flight software as of February, 1997 is shown in Figure 1. A high degree of spacecraftautonomy is made possible by a run-time architecture consisting of: planner/scheduler, mission manager, smart executive, mode identification and reconfiguration, planning experts, fault monitors and real-time execution. The real-time execution segment is where the traditional real-time flight software such as attitude control resides. The planner/scheduler, mission manager, the smart executive and the mode identification and reconfiguration (also referred to as fault diagnosis and recovery) work closely together and are known collectively as the Remote Agent. This is the top application layer of the flight software managing the operations of the spacecraft functions and it is the core of our new autonomy architecture.

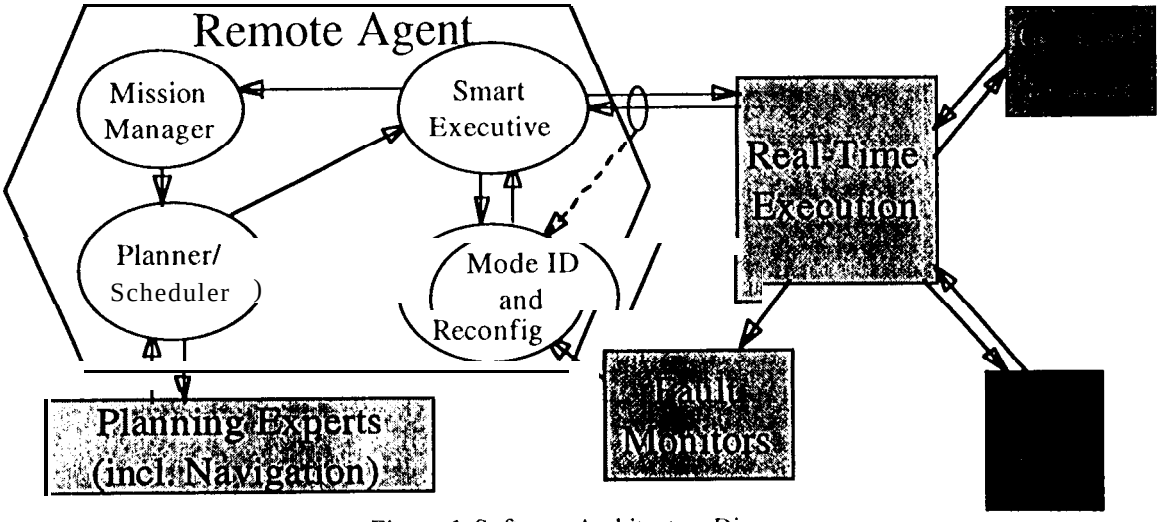


Figure 1. Software Architecture Diagram

The Remote Agent's design [3] is based on powerful and exciting technologies which endow the spacecraft with new behavior and capabilities. The Remote Agent carries explicit models of the operational behaviors of the spacecraft such as hardware state transition models, resource requirements, and operational constraints. The reasoning engines in the Remote Agent look for conflict-freeprocedures cooperate the spacecraftin real-time. The benefit of this approach is to bestow the spacecraft with multiple means to function in uncertain environments while reducing mission operations uplink requirements. Constraint-based planning and scheduling ensures achievement of long-term mission objectives and manages the allocation of system resources. The smart executive performs robust, multi-threaded execution to reliably execute planned sequences under conditions of uncertainty, to rapidly respond to unexpected events such as component failures, and to manage concurrent real-time activities. Mode identification and reconfiguration, based on model-based diagnosis, uses hardware models to infer the health of all system components based on inherently limited sensor information and to generate novel repair sequences. The three engines work together to transform the spacecraftfrom an open loop-system (as seen by the ground operator) to a closed-loop system. The closed-loop approach allows the spacecraft to go into environments that are more uncertain and to achieve our science goals more reliably in the face of such uncertainty. The other potential benefits of the Remote Agent are to reduce mission development cost through software reuse, to provide faster response to in-flight problems or opportunities, and to reduce **operations** cost.

The Remote Agent also provides a scaleable, modular architecture for flight software. Specialized functions are delivered as custom modular domain experts. These functions can be traditional spacecraft subsystem functions or they can be domain-specific autonomous functions incorporating new technology, such as optical navigation. This flexible and scaleable architecture can serve a wide variety of systems, both in space and on the ground,

DS1 flight software had approximately 120,000 lines of code and 50% of it belonged to the Remote Agent. The Remote Agent was part of the primary software. There were two "rate groups": 1 Hz and 1 to 0.1 Hz. The Remote Agent was in the l-to-O. l Hz rate group and it did not have hard deadlines. Figure 2, an interface block diagram of the flight software, shows all the software objects. The DS 1 flight hardware is accessed through the 1553 bus object as shown in the figure.

4. Flight Software Implementation Approach

The RA was selected as one of the technologies to be validated on DS 1 because it promised a significant impact on 21 st century science missions and because it needed flight validation to reduce risk. The autonomy team consisted of autonomy software technologists from the NASA Ames Research Center (ARC), the Jet Propulsion Laboratory (JPL), Carnegie Mellon University (CMU), I'RW Spacecraft Technology Division, and spacecraft system and mission engineers from JPL.

Autonomy software technologists bring a host of techniques including inference and search engines, multi-tasking capabilities, sophisticated error correction, model-based reasoning and declarative modeling languages.

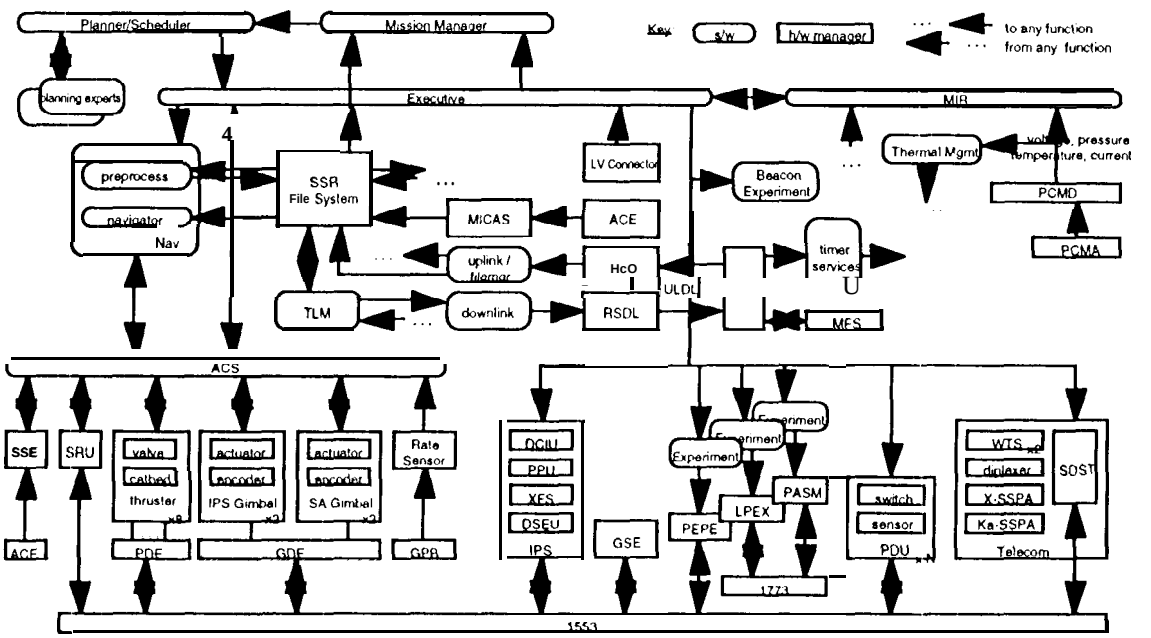


Figure 2. DS I Flight Software Block Diagram

These techniques can be used to develop onboard software capable of conducting spacecraft mission activities with much reduced reliance on ground intervention. On DS1, software researchers were also implementers and worked closely with spacecraft engineers to implement the Remote Agent. The intent was forthe spacecraft engineers to learn these methods and understand their applicability and limitations from the technologists. For future missions, the spacecraft engineers will implement the Remote Agent with the technologists acting as consultants but not directly involved.

In preparation for submitting the Remote Agent as a candidate for flight validation, a rapid prototype was developed to demonstrate the feasibility and applicability of the system. Known as the New Millennium Autonomy Architecture rapid Prototype (NewMAAP), the prototype demonstrated the autonomous execution of a simplified version of the Saturn orbit-insertion activity for the Cassini mission [2].

To reduce implementation risk, in addition to the normal project reviews such as preliminary design review and critical design review, the NMP technologies were required to successfully pass three readiness gates:

- Gate 1: show maturity and an adequate and complete flight implementation plan.
- Gate 2: demonstrate functionality at the subsystem level.
- Gate 3: demonstrate functionality at the system level.

The DS1 project was given a doubly challenging task: not only was it required to demonstrate a record number of new technologies, but it was asked to complete the implementation in a short schedule of only 2.5 years from inception to launch without even the benefit of a preproject phase A. In order to meet the short schedule and develop a software design concurrently with hardware developments, the flight soft ware team adopted a spiral implemental ion process as opposed to the traditional waterfall approach. The software development process was divided into 6 cycles, RI through R6. Each cycle releases software of increased functionality and by the final release the software has complete functionality capable of executing the full mission. The capability in each cycle was identified by a scenario defining a mission segment. Each software component correspondingly develops the functionality to accomplish the activities required by the segment.

Figure 3 contrasts the conventional waterfall method with the spiral approach. Figure 4 identifies the mission segments for each release.

The advantages of a spiral implementation plan are:

- It provides early feedback on interfaces and system architecture.
- It allows early integration of all components including ground and test environment.
- It makes an end-to-end running system available for system-level tests.
- It enables test tools and procedures to be identified.
- It facilitates incremental deliveries.

Since DS 1 had no preproject phase A, little experience was available to guide the interval and work load between releases. Experience from the previous release was used to guide the planning of subsequent releases. Figure 5 shows the schedule forth; spiral development approach.

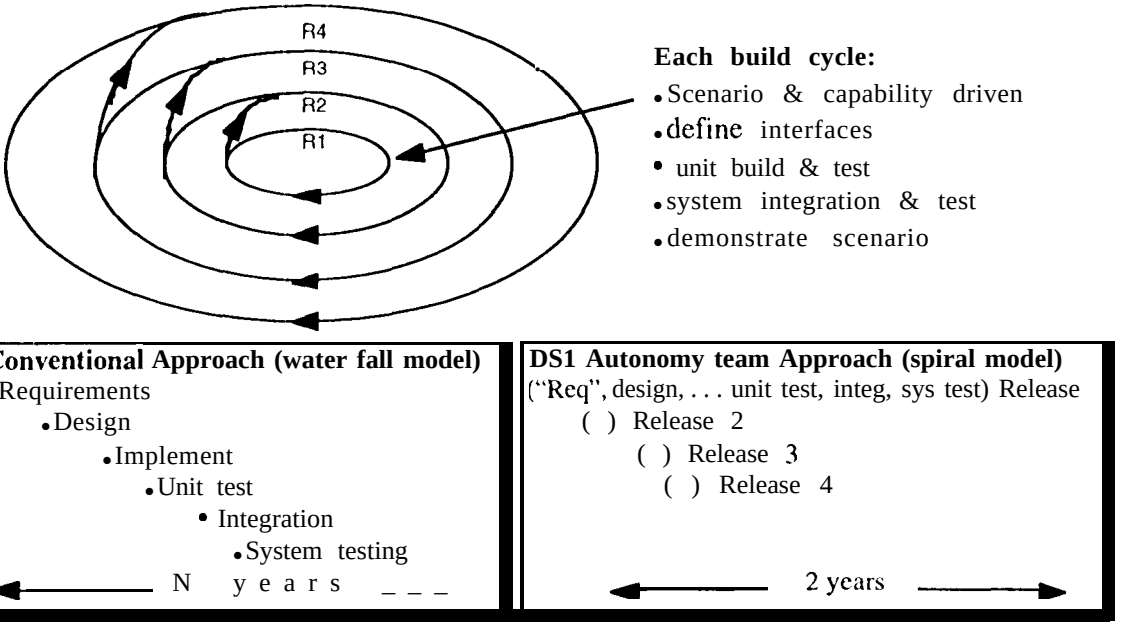


Figure 3. The Spiral Development Process

RI: Ballistic Cruise scenario on workstation environment with

- Inter-Process Communication,
- " Remote Agent,
- " Optical-Navigation orbit determination
- Preliminary Attitude Control Subsystem (ACS).

R2: Add Ion Propulsion System (IPS) cruise & Earth Communication scenarios; on mixed VxWorks (68040)/workstation environment with

RI capabilities +

- MICAS imaging for navigation maneuvers,
- Reaction Control System (RCS) attitude control,
- Command and Telemetry, Beacon,
- Ground segment data transport.

R3: Add Flyby scenarios on VxWorks (PPC/ Commercial RAD6000) environment* with

R2 capabilities +

- Autonomous navigation,
- RCS delta-V,
- Solar array controller,
- Ground segment command/telemetry,
- Telecom system including Small Deep Space Transponder,
- Power system incl. SCARLET and battery,
- MICAS science.
- Planner still on orkstation

R4: Add Power-On-Reset and Launch scenarios

R3 capabilities +

- ACS with cruise, IPS, panel control,
- IPS variable thrusting,
- IPS diagnostics package,
- Planner bypass,
- " Thermal control,
- PEPE science.

R5: **Add Technology experiments.**

R4 capabilities +

- SCARLET characterization.

R6: Update R5.

- Validation continued.

Figure 4. Software Releases, RI -R6 Definition

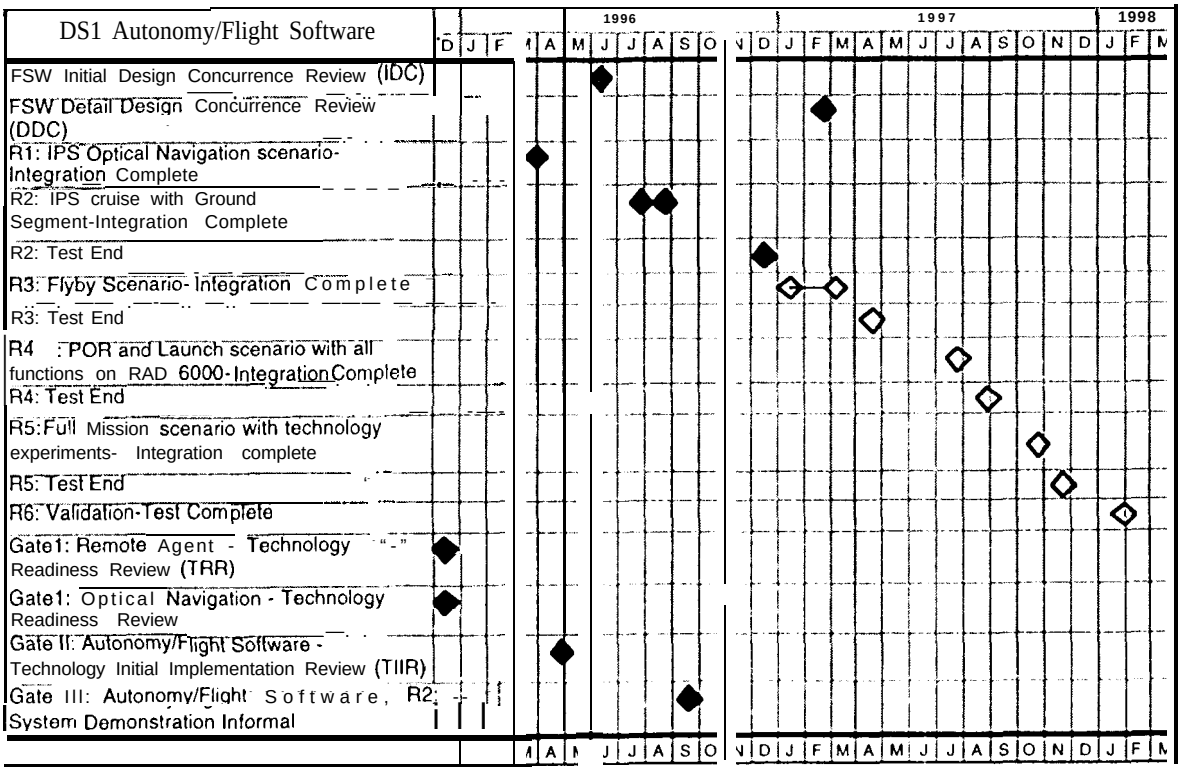


Figure 5. Autonomy/Flight Software Schedule

5. CHALLENGING INITIAL CONDITIONS

The preceding sections have already discussed some of the conditions and constraints which framed our technology infusion effort. By highlighting some of the key conditions in this section, we set the stage for many of the lessons which follow. Our discussion of initial conditions and assumptions is organized into two major themes: adoption and schedule.

Technology Adoption

Technology validation can raise particular challenges for marketing and customer acceptance, because it relies on a community of users to serve as "early adopters". These early adopters accept a fair amount of risk, because they must work with a technology which has not yet been demonstrated and accepted as standard in the ultimate operational context. In the case of autonomy for DS 1, the ground data system (GDS) and mission operations system (MOS) personnel were the early adopters, who would use the remote agent. However, the following aspects were challenges from the start:

First, there was limited motivation for the GDS or MOS participants in DS 1 to embrace the Remote Agent technology. While it was recognized by many that Remote Agent's autonomy capability was an important enabling technology for future missions, it was not actually an enabling technology for DS1. All mission phases for DS 1 could be accomplished with traditional technologies that the ground team was familiar with. Moreover, unlike some of the other technologies to be demonstrated on DS 1, the Remote Agent was not listed as a **required** or mission-defming technology on this mission. Instead, it was treated as a goal, which was part of the baseline but could be dropped if the project encountered more pressing concerns.

In order to focus DS1 participants on making the technologies work, the project was organized in such a way as to depend on them despite the perceived risks. The program office set the goal of developing "no backups" or alternatives to the Remote Agent technology, and the ground operations budget was restricted such that the ground team would have to take advantage of the autonomous capabilities to reduce operations costs. Similar constraints were placed on the flight software development team. There was to be no safety net or backup fault protection to enable the mission to be run if the Remote Agent failed for some reason. In effect, the Remote Agent was placed both on the critical path of the mission and in a mission-critical position within both the flight software and the ground operations capabilities.

There were also differing opinions about the ultimate purpose of the new agent technology for future missions. The primary intended use of autonomy is to leverage the mission controller efforts by allowing the spacecraft to function as an extension of the mission controller's capability. In contrast, some pre-DS 1 autonomy technology advocacy focused on the concept of autonomy enabling "no uplink" missions (perhaps eliminating the need for a receiver on the spacecraft), in which the spacecraft achieved its mission and sent down information to ground without *any* input from ground during flight. While uplink may be limited or impossible for some missions, support for such missions is not the main benefit of autonomy. Focusing on this extreme context neglects many of the needs perceived by the mission operations community. In particular, it avoids discussion of a meaningful interface for ground operators to interact with the spacecraft. Enabling operators to interact at whatever level is most appropriate and feasible is a more desirable goal for autonomy [4].

Schedule and Budget

The other major set of constraints on the mission had to do with schedule and budget. These have been referred to earlier. The fast schedule and limited budget were more aggressive than those of any previous project. Thus, the team was asked to develop the processes required for a new software approach and execute those processes in less time than was typically allotted to traditional flight software development. Also, the hardware much of it also new-technology development was to be developed concurrently with the software. These conditions implied that we would learn about new requirements and interactions throughout the project, with limited time to respond to them to everyone's satisfaction. Combining this with the context that we were inserting a new technology which was at the same-time mission critical, it was difficult to separate the schedule slips caused by the aggressive schedule from those related to the technology development. See Yourdon [5] for a discussion of the impact of initial conditions on software development projects.

In an effort to meet the tight schedule and budget constraints we used a spiral flight software development approach. While the RA team supported the spiral development approach, spiral development is not a necessary part of the RA design. The lessons teamed from this development approach are included in subsection **7.11** here to give the context in which the technology was being inserted.

As should be clear, the initial conditions were very challenging. Ideally, the Remote Agent would have been directly linked to mission success, rather than indirectly as it happened. Such direct linkage could have been in the context ofa mission which required the agent capabilities as mission enabling (for example, a comet lander such as DS4). In the absence of this, additional schedule and resources could have increased chances of success.

6. ROLE REDUCTION OF THE REMOTE AGENT

In March, 1997, the Remote Agent status on the flight changed, Instead of being the software responsible for guiding the spacecraft's flight operations, it is now being flown as an experiment. The decision to change its status was made based on difficulties encountered by engineers in the integration and testing stage of the flight software development process. The development of the integrated software was not expected to reach a stage of "flight readiness" to meet DS I's delivery schedule for launch. The

difficulty was compounded by the perceived dependence of the Remote Agent on the 3-D stack computer, which is another new technology. The 3-D stack also had major schedule difficulties. Due to schedule and integration difficulties, the 3-D stack computer was replaced by the RAD6000 computer, similar to the one used on the Mars Pathfinder mission; the Mars Pathtinder software forms the basis for much of the DS1 software.

The development of the Remote Agent software, however, was going well. In fact, based on the work done, the team developing the technology had demonstrated that it is every bit as exciting and promising as we first believed. While we will not have the complete capability that this software provides on board the DS 1 spacecraft at launch, as originally planned, NMP will still continue to develop the technology so that the capability can be uplinked to the spacecraft after launch. The Remote Agent will be tested in space as a week-long experiment [3].

The autonomy capability development and testing task should be thought of as a continuous effort being undertaken by the NMP; whatever portion of the technology is not flown on one advanced technology flight (e.g. DS 1) will be flown on another flight and so on, until the full technological capability has been tested in space. To this end, we are planning for the final portion of the DS1 Remote Agent software capability to be tested on the Deep Space 3 (DS3) mission.

The change in status of the Remote Agent software technology does not reflect a lack of achievement by its development team. On the contrary, the team's effort has been outstanding. The reason this promising technology's status was changed is because it was a part of a flight software development that did not meet the ambitious development schedule set by NMP, and the Project Manager saw a possible way to reduce overall project schedule risk by removing the new technology from the critical path. Though NMP's goal is the aggressive pursuit of developing and flight-testing revolutionary, high-risk technologies, it recognizes it may sometimes be necessary for projects to back off from some technologies if they are on the project's critical path. The main body of this paper captures lessons from development of the Remote Agent from October, 1995 (project start) to March, 1997. We intentionally omit lessons learned by members of the project outside the autonomy team.

7. Lessons Learned

7.1 Impact On Project

Observation

The magnitude of the impact of the Remote Agent on the project development process surprised both autonomy and non-autonomy people alike.

Discussion

The Remote Agent approach required increased levels of knowledge capture and formalization earlier in the process than in traditional missions, In order for the Remote Agent to generate and execute a viable sequence to accomplish a desired goal, it uses a large body of knowledge stored onboard. This knowledge consists of:

- the behavior of the spacecraft as designed by the system and subsystem engineers.
- the behavior of the spacecraft hardware and its interaction with the outside world, both nominal and off-nominal.
- the operation of the mission including flight rules and constraints as developed by the mission design team in conjunction with the project as a whole.

Arguably the same amount of knowledge and understanding is required by a traditional approach. However, there are significant differencesin terms of timing and precision of knowledge capture required for autonomy. First, our approach required substantially more of the system to be formally captured in software models. Second, the models represented both nominal and fault behaviors. Whereas traditional missions leave much of the failure analysis to be performed post-failure, our approach facilitated increased robustness at the cost of up-front modeling of explicit failures. That way, the system could diagnose and respond to more problems onboard. Third, the information was needed much earlier in the development process than in a traditional approach. As in the case of fault models, our approach enables more of the mission operations to be worked out on-board the spacecraft, but our approach required that flight rules and constraints be encoded within the flight software development process, rather than as part of the ground system development process. Final ly, the information needed to be transferred across multiple software components as part of the original design, whereas the traditional approach involves people talking across organizational boundaries much later in the process. On recognizing the demand for knowledge management, a hardware modeling team was formed to coordinate the hardware knowledge and develop the models.

In addition to the increased requirements for knowledge management, system-level autonomy software influences and is influenced by virtually the whole project, Thus the infusion of system-level autonomy differs from the infusion of new hardware technologies, for example a new processor, where interaction with the rest of the system is limited. System-1evel autonomy affects the spacecraft system, the ground system, and the mission operations system. Consequently a large number of people across the project needed to understand the changes that autonomy introduces and its effect on their work. Big impacts were on the system engineering, ground/operations, and the system-level test teams. For example, the test team needed to allocate additional resources to account for their understanding of the autonomy system and to develop test-plans to give them the same confidence and comfort level of system reliability that they traditionally provide. Moreover, increased autonomous capabilities created more software and more required test cases, further compounding challenges already present in testingspacecraft software.

The understanding of' the magnitude of the impact of autonomy on the project evolved with project maturity and in some cases, as in the case of "modeling demands," plans were made that addressed the issue. Other cases, which surfaced later, such as additional resources for system-level tests, could not be accommodated principally because DS 1 operates under cost caps and a tight schedule.

Lessons

Technology readiness, while necessary, is not a sufficient indicator for the successful infusion of new levels of autonomy in missions. An assessment of the impact of autonomy on the development of the rest of the system should be made.

Recognizing the significant impact of autonomy on project development, the technologists should make recommendations to the project on specific adjustments in the development process and project organization to facilitate communication, visibility and teaming.

The ground/operations and system-level test teams should be included in the autonomous system prototype development to reduce the learning-time burden during a flight project. Alternatively, sufficient resources and margin, especially schedule margin, should be allocated when including autonomy to counter the effects of uncertainty on the whole implementation process. On 1)S 1 autonomy amounted to a "revolutionary" change in the development and content of the flight software and ground systems.

Again, it should be noted that the rapid schedule of DS 1 eliminated the option of conducting a "Phase A" study. We highly recommend such studies in the future, especially in cases where new autonomy capability is to be used, as this can help identify many of the system-level interactions and promote a better-informed schedule and process.

7.2 **Launch Readiness**

observation

Autonomy implementation was in the critical path and was perceived to threaten the progress of the project to make the launch date.

Discussion

On DS 1 the implementation of autonomy was not a software layer built upon a traditional non-autonomous system. Rather the architecture consisted of three, tightly interacting autonomy modules, the Smart Executive, Mode Identification and Reconfiguration (MIR), and Planner/Scheduler (P/S). The Smart Executive subsumed the role of the on-board sequencer and could read both plans and sequences from the ground. However, it did not handle the sequence blocks using the structure and tools previously used by the ground system.

MIR embodied the basics of the traditional fault-protection function, albeit of much greater capability, of diagnosing equipment failure and providing a recovery path or reverting to stand-by when the failure is beyond onboard recovery. As part of the Remote Agent, MIR informs the Executive of the spacecraft current state and confirms to the Executive when commanded activities are accomplished. The overall-fault protection function was accomplished by all three modules working in concert.

The Planner/Scheduler module subsumes the traditional ground operator task of generating plans. The module schedules the desired activities with the help of domain-specific planning experts that estimate the required resources and check for constraint violations. The plans generated are high-level and are implemented by the Executive.

Of the three modules only the P/S could be located on the ground, allowing additional human inspection of the plans before they were sent to the spacecraft. Indeed this option was available on DS I and known as the planner-bypass mode. It was possible with the Remote Agent architecture to reduce the extent of autonomy, even drastically. However, with this architecture it was not possible to totally bypass the RA; some parts of the RA were essential to operate the spacecraft. This dependence on a new technology raised enough of a concern that eventually, when integration problems were threatening the launch schedule, the project reduced the role of the RA to an experiment, removing it from the critical path.

Lessons

When autonomy is given an opportunity to be part of a mission, every effort must be taken to first enable launch capability. During implementation we must provide early validation of any new technology on the critical path. We must jointly with the mission operations team demonstrate early-on the capability of the new technology to perform the traditional tasks that it is replacing so as to avoid a perception that the technology is a risky item on the critical path.

Ideal ly, the chosen architecture should make autonomy transparent to the traditional approach and allow for various levels of autonomy to be implemented incrementally. This approach is supported in the newer version of the RA [4]. Management should maintain detailed contingency plans for juggling scope and schedule. They should carefully track the progress and be prepared to invoke the contingency plans when problems surface so as to always safeguard launch readiness. Special care should be taken when the autonomy technology is being demonstrated on a mission for the first time.

7.3 **Impact on GDS and MOS**

observation

The effect of the Remote Agent on the requirements and design of the supporting Ground Data System (GDS) and Mission Operations System (MOS) was unclear to the developers of those systems.

Discussion

A traditional GDS includes models of the spacecraft's hardware and software sufficient to predict, in response to a

proposed sequence of spacecraftcommands, the state of the spacecraft and how it varies with time. The level of detail of the predictions (and hence of the models) is principally determined by the flight rules that every sequence needs to be checked against. Also, spacecraft subsystem analysts require predictions sufficiently detailed that comparisons of predicted values to actual values enable them to determine how their subsystem is behaving.

The developers of the GDS felt caught between two conflicting views. In a mature, intelligent spacecraft, one that is commanded by high-level goals, and that can be trusted to carry out reliably the necessary actions to accomplish the goals, only high-level, simple models would be needed on the ground to prepare a sequence. (Even this statement skirts the issue of how to know that a set of goals is accomplishable.) This view implies that developing the GDS would be simpler than in a traditional project.

The conflicting view saw a first-attempt, pathfinding effort at on-board intelligence. For diagnosis of inevitable surprises, this view saw the traditional level of predictions as necessary as on a traditional mission. What's more, such predictions are more difficult to make, because the models that make them would need to include models of the on-board intelligence. Indeed, unlike in traditional missions, detailed command-level predictions with accurate times are simply not possible, since the Remote Agent uses actual spacecraft states unknown to the ground predictors.

An additional unease felt by the GDS and MOS developers stems from the fact that, commonly on projects, the flight software at time of launch has fewer features than originally planned (for many different reasons, including realities of memory and processor speed). It is the traditional role of the GDS and MOS to make up for such deficiencies. Contemplating the possibility of an "unfinished" Remote Agent brought fear of failure of the MOS to be able to fill the gaps.

So there was a dilemma. The (traditionally, at least) healthy skepticism concerning whether the flight software would do all that it originally intended to do was mixed with outright skepticism of the whole autonomy effort.

It was not clear to the developers of the GDS and MOS which traditional functions of those systems would be supplanted by spacecraft features: which needed to be modified from tradition, and which would remain traditional. These developers feared that the autonomy developers were inadvertently inaccurate in their descriptions of effectson the GDS and MOS from lack of experience in those areas. Yet the people with that experience were not knowledgeable on the autonomy, and so could not make their own assessment of effects on the GDS and MOS.

Lessons

The effectof the introduction of autonomy in one part of a large system on other parts must be gauged not only under the assumption of complete success of the autonomous element, but also under realistic possibilities of shortfalls.

The autonomy technologists and traditional GDS and MOS developers must jointly and explicitly construct a mapping of the traditional ground-supplied operations functions to the corresponding ways of accomplishing those functions in the new, autonomy-centered system.

An incremental approach to autonomy, which lets GDS and MOS developers develop and exploit more autonomy capabilities as they gain confidence in the technology, may help ensure that important capabilities supported in the old approach are still supported by the new technology.

7.4 Concurrent Development of Autonomy, CDS and MOS

Observation

The autonomy development team, rather than the MOS developers, prepared mission scenarios

Discussion

The spacecraft must understand high-level goals. this implies that knowledge for deriving (from a goal) the low-level commands needed to effect the goal must be rigorously codified as part of the development of the spacecraft. Also, as mentioned in section 4, the approach for phasing the software development was based on mission scenarios. For both of these reasons, it was the flight software development team that took the initiative in devising the mission scenarios and their details. Devising scenarios was done concurrently with cycles of the software development, as a way of tinding requirements and design for the on-board software.

In contrast, in a traditional mission with a longer development period, development of sequences is not attempted until the low-level commands are defined. The development of blocks, which are reusable, parameterized sequence fragments, is led by the spacecraft system engineers. The implementation of those blocks in ground software is the responsibility of the GDS developers. The development of most sequences of commands and invocations of blocks is the responsibility of the MOS developers.

The introduction of autonomy, and short development time with or without autonomy, both argue for concurrent development of MOS, GDS, and flight software. The roles of the various developers of these systems thus must change.

Also, because of the concurrency in development, changing details of the codification of a goal in the on-board software should be easy mechanically, and doable without changing unaffected goals orothersoftware elements.

Lessons

Specifying details of how to operate the spacecraft to accomplish goals that are intended to be understood by the on-board autonomy must be done early enough to be encoded in flight software.

Such details will influence the flight software capabilities and commands, and therefore must be done concurrently with the development of the flight software.

MOS and GDS developers must work concurrently with flight software and autonomy developers in defining mission scenarios.

7.5 Teaming of Autonomy and GDS/MOS Developers: Differing Perspective

Observation

Though the autonomy team spent considerable effort informing MOS developers of intended spacecraft features, there remained a separation between the two groups of people.

Discussion

On DS 1, there was close cooperation between autonomy personnel and MOS developers. Weekly MOS Design meetings were held, with active participation by autonomy personnel. Yet, as launch drew nearer and details needed to be worked, a gulf between the two groups seemed to appear.

This experience has multiple roots. One aspect is communication. The MOS has a terminology built up over many missions. As in any specialty, what seem to a newcomer to be ordinary English words in fact bring with them a rich circle of associations. These extra meanings are tacitly understood by MOS experts, while non-experts may not even know that such associations exist. Conversely, autonomy researchers have their own terminology not understood by outsiders such as MOS developers

Surprisingly, the confusion continued, perhaps because in addition to using differentterminology, the groups did not share a single paradigm. The paradigm of experienced MOS developers includes a strong emphasis on risk avoidance. No sequence of commands to the spacecraft should cause the spacecraftto exercise its fault protection capabilities; MOS personnel consider such a sequence a failure. Every detail of the operation of the spacecraft by a sequence should be predicted and validated. If anything deviates from the plan, this is viewed as a potential emergency and the spacecraft should be put into a safe state from which MOS personnel can respond.

The autonomy approach enables the spacecraft to respond to some of these abnormal situations by taking a recovery action and generating a plan appropriate to the new situation. This means that MOS personnel will not be involved in some cases where they normally would have been. Until they are convinced that the autonomous approach *really works*, they will feel very uncomfortable about giving up any of their protective role in spacecraft operations. In the case of a newly developed autonomy capability, MOS will not have confidence until very late in the design process (possibly not even until a month or so into flight). Thus they will be on their guard and extremely conservative about what capabilities are delegated to the autonomy software.

Another difference between the groups is the primary goal of each group. This difference pertains to any new technology. The technologists' fundamental goal is to demonstrate that their technology works (which requires a successfully executed mission). The primary goal of the MOS personnel is to execute the mission. The analogous tension in a traditional project is that between the scientists and MOS personnel. At times, the scientists must be reminded that no observations will come forth if the spacecraft is injured. Conversely, at times the MOS personnel must be reminded that having a safe spacecraftbut making only a few science observations is missing the point of the mission.

Lessons

The MOS developers must actively participate in a shared vision of the role of autonomy on the mission. MOS developers and autonomy developers must form a single team. Working together, they must develop and use a single set of terms. One way to effective team-building is to jointly work on a prototype before the implementation of a flight project with its attendant tight schedule.

Realistic operations scenarios must be worked through to bring out differences in viewpoint. Resolution of the differences must be agreed to.

7.6 "Last-Minute" Fine-Tuning of Sequences Before Committing to Execution

Observation

Traditional "last-minute" tine-tuning and checking of sequences of commands, often related to up-to-date knowledge of the spacecraft's trajectory, is not applicable for a spacecraft with autonomy.

Discussion

In a traditional MOS, for an activity that will be repeated during the mission, a "block" is defined. A block is a parametrized set of spacecraft commands, with time spacing, that will accomplish the activity. Specific instances of the blocks are tested before launch. Later, nearer the time of intended execution, each sequence, consisting of instances of blocks plus some individual commands, will be built and tested. In some cases, minor changes will be made to the sequence either just before or even after the sequence is uplinked to the spacecraft(but before the scheduled time of the first command in the sequence).

The testing and analysis of blocks is meant to cover all reasonable values of the parameters in the blocks. But there is a significant sense in which such generality is not completely depended upon. Namely, each specific sequence always goes through some level of testing, which covers some aspects of the total set of commands in the sequence, even those that come from a block. Traditionalists are used to doing some testing and human review of every specific sequence that the spacecraft will execute.

In an autonomous spacecraft, the role of people on the ground is necessarily changed. It is the spacecraftsoftware

that does the "last-minute" tine-tuning; on DS 1, the autonomous navigation system updates knowledge of the spacecraft's trajectory. Humans never see the actual specific instance of the sequence of commands before execution. Humans must trust the software to work in the general case; there is no final check by humans of the specific sequence.

The presumption on which autonomy developers depend is that the closed-loop nature of the autonomy more than makes up for the lack of final human inspection of each specific sequence. Instead of the traditional job of constructing an open-loop sequence that will accomplish the desired results even in the face of some unknowns, the job is one of building models sufficiently correct that the autonomous engines, with the advantage of real-time knowledge, can accomplish the desired results.

Lessons

A large challenge is for the autonomy developers to convince themselves and others that the autonomous software will work in the general case. It is paramount that a validation strategy be used to ensure that the models on which the autonomous software depends are complete and correct enough to succeed,

MOS personnel must become comfortable with planning and monitoring at a higher level than traditional. In addition, autonomous software systems should support the ability for ground to interact at multiple levels of detail, so that they can revert to a traditional level when necessary, perhaps varying their level of interaction on an activity-by-activity basis.

Many members of the ground operations community would prefer for the introduction of autonomy to be able to be phased, converting more functions to autonomous versions as experience is gained on previous functions.

7.7 **Validation Before Flight**

Observation

There was constant concern about verifying and validating autonomy, particularly within the system test team. There was a misperception that the RA was a non-deterministic system and that it therefore could not be adequately tested. there was also the accurate perception that the increased complexity of the software required a very large testing effort.

Discussion

A key issue that troubled the system test team was predictability. A traditional sequence-driven flight software system is very predictable; the test team can set the initial conditions, run a sequence, and observe the results at a detailed level. The Remote Agent, in contrast, is an onboard closed-loop control system that reacts to onboard events and spacecraft-state, i.e., to data that is not a priori predictable.

This unpredictability of onboard conditions, and thus unpredictability of RA behavior, led to a misperception that the RA was non-deterministic. In fact, the RA is deterministic to the extent that the same set of inputs will yield the same set of outputs every time. However, it is true that we cannot predict the exact set of commands that the RA will use to achieve a set of goals far in the future since we cannot predict exactly what the spacecraft state will be at that time.

The key to testing such an "unpredictable" system can be found in another context, namely, attitude control systems, While we don't know exactly when a particular thruster will fire, we do know that the system will fire thrusters as needed to achieve the higher-level goal of holding a commanded attitude. Such a system is tested under multiple scenarios to ensure that pointing error requirements are respected and that propellant usage is acceptable. Similarly, the RA must be tested under a variety of scenarios while verifying that goals are achieved, flight rules are respected, and resource usage is acceptable [3].

Lessons

It is incumbent upon autonomy technology providers to codify a strategy for system-level verification and validation, and then communicate that to the system test team. This communication must build from a shared understanding of the RA as a deterministic control system whose behavior must be verified to remain within specific behavior envelopes over a space of initial conditions and external perturbations. Exhaustive testing is not feasible; what is needed is intelligent spot-checking of a variety of nominal situations, boundary conditions, and stress tests.

7.8 Validation In Flight

Observation

It is tricky to design in-flight validation scenarios to test some of the capabilities of autonomous software systems.

Discussion

The primary reason for flying RA on DS- 1 was to validate the technology for use in future missions. Some of the RA's capabilities, like advanced fault protection, are only demonstrated in-flight if a problem arises. However, it is difficult to see how to inject such problems in-flight without endangering the spacecraftand the mission. Such difficulties meant the RA flight validation scenarios were not fleshed out until late in the design and it was difficult to obtain resources (such as downlink capability) to support the demonstrations. Similarly, some of the proposed benefits of the RA, such as reduced operations costs or increased software reuse, are hard to measure if there is no separate operations process to compare it to during the same mission.

Lessons

Autonomy technology developers and management should carefully evaluate the tests which need to be conducted in-flight as soon as possible and develop with the project a shared understanding of how the flight tests will be

conducted. The technology demonstration should also accept that not everything will be tested, and that ground-based testing will form an integral part of the technology demonstration process. The software itself should be designed with flight and ground-based testing needs in mind from the start.

Autonomy development projects should set up processes to carefully measure the benefits of the technology. This may require increased investment of resources to support side-by-side comparisons. Montemerlo has argued for the importance of such comparisons, which he calls the "John Henry test", in convincing future customers to adopt a new technology [6].

7.9 Architecture

Observation

DS 1's flight software architecture cleanly separated concerns of real-time control, deduction of spacecraftstate, executive control over multiple concurrent threads, and goal-based commanding via onboard planning. Further, a significant amount of the "programming" was really knowledge engineering, i.e., expressing spacecraft knowledge in specialized non-procedural languages.

Discussion

The objective of autonomous operation for DS1 required an onboard closed-loop approach that included "knowledgeable" components for performing much of the reasoning that heretofore had been performed by ground operations. This requirement led to the specialized **reasoning** engines of the Remote Agent— engines for planning of activities, for multi-threaded execution of plans, and for fault localization and recovery. Although there are important and necessary interactions among these engines, they provide a clean separation of concerns in the software architecture that mirrors, to some extent, the specializations seen in traditional ground operations.

The RA engines each provide a distinct kind of reasoning, and thus place different demands on how they are programmed. For example, the planner/scheduler needs to know the dependencies and constraints among activities for accomplishing goals, and the mode identification/reconfiguration engine needs to know how subsystems behave, in nominal operating mode as well as in failure modes. Each engine was "programmed" with such factual knowledge using a non-procedural modeling language designed for the task. The non-procedural nature of these languages, plus the inference engines behind them, allowed developers to focus on the easier task of stating *what* is known versus the harder task of stating *how/when/where* to apply it.

The organizing principles for control and knowledge provided by the RA architecture gives programmers considerable guidance and structure in its implementation. Without it, as in more conventional spacecraft when the target implementation language is a procedural language like C, there is a much higher likelihood of software that intertwines different issues in unprincipled ways.

Lessons

The basic design was right. The three major components of the Remote Agent plus the real-time control component provided a clean separation of concerns that mapped into an agreeable division of labor. The existence of the RA's specialized inference engines, plus the declarative languages for programming them, brought a lot of leverage that made a difficult task more tractable.

7.10 Autocode Generation

Observation

Code generation from formal specifications was a big win, and even bigger when the specifications included acceptance tests.

Discussion

In DS1 there were a number of ''monitor" modules whose job was to notify the RA in the event of a change in state of various measurements such as switch positions and discretized low/nominal/high sensor readings like solar array voltage. As the responsibilities of monitor modules evolved to include things like telemetry recording and parameter updating, it quickly became a tedious process to update all monitors or even confirm that they performed a given task in a consistent way.

The structural similarity of the monitor modules suggested that a code template could be tailored for individual monitors given a terse specification of an individual monitor's requirements. Further, since it was clear that the person who would specify monitor requirements would be in the best position to specify acceptance tests, the specification language was expanded to include such tests.

Although the effort in building the code generator and code template was substantial, especially in the middle of a development cycle, the payoff was well worth it. Not only did it automate a tedious coding/testing job and ensure a consistent product, it also mentally freed the monitor engineer to focus more on the problem (measurements, noise filtering, thresholds, etc.) without concern about code design and development, This really transformed a tedious job into a pleasurable activity and yielded a tool that can be reused on future projects.

Lessons

When a design contains structurally-similar instantiations of a non-trivial computational task, seriously consider code generation from a formal specification. Even ignoring the gain in productivity from code generation, there's real value in the clarity that comes from thinking in terms of the problem rather than the implementation, and there's comfort from the guaranteed consistency.

7.1 I Spiral Development Approach

observation

Difficulties were experienced with the cyclical incremental deliveries of the spiral development process.

Discussion

As describedin section 4, the spiral method of the flight software development process resulted in cyclical incremental releases. This process allowed the early development of an end-to-end functioning system that included all the subsystems, albeit of low fidelity in the early cycles, complete with the ground system and test environment. The intent was to demonstrate the scenario with an end-to-end system, commanding through the ground system and assessing the system and subsystem behavior in the testbed.

In our implementation of the spiral development method we required all subsystems to make a delivery of increased capability at each cycle. This uniform requirement of all subsystems to deliver at every cycle proved to be troublesome in two ways. First, integrating all the software modules in one "big bang" made the integration process very complex, required extensive coordination to debug, and consequently took longer than planned to complete.

Second, some subsystems whose natural progression of increased functionality required a longer development period than the cycle interval, found it difficult to juggle long-term planning to complete the module and the short-term requirement to deliver for the initial cyclical releases. For example, as shown in Figure 4, the attitude control functionality was scheduled for R2, but would have been better scheduled for R3. It takes more than one cycle interval to develop control laws and to integrate them with the attitude control subsystem. From the beginning we should have planned that some modules would skip certain releases in keeping with their development pace and to ease the integration process.

Another development mistake we made in an effort to meet the extremely short schedule was to overlap the cycles. That is, the next cycle was started when the previous cycle completed integration but was just starting test. The overlap required each of the development teams to work concurrently on the testing of the previous cycle and the development of the next release. With staff on some teams already stretched thin, this concurrency led to the testing of each cycle to be shortchanged.

[n hindsight we could also have ordered the scenarios better. For example, we scheduled launch power-on-reset and system initialization for R4 when it would have better served us to schedule it in R3. The initialization capability was needed in the integration and test of the R3 release which had already incorporated the major functional ities needed by the full system.

While the spiral method affords flexibility to change contents from one cycle to another as warranted by the development circumstances, there is a danger of continually deferring contents into future cycles thus giving an inaccurate assessment of progress. On DS I during planning we allocated R5 as a contingency against such deferment and we devoted R6 to validation and rework,

Lessons

Care should be taken when developing the schedule for incremental system level releases not to overburden the integration process by requiring all software modules to deliver at every cycle.

The subsystem developers' own assessment of a doable increment not only to be delivered on time but which they can support during test must be taken into account when developing the spiral method plan. New releases do not necessarily warrant updates from all subsystems.

Each cycle should be completed, including all testing before, proceeding to the next cycle.

Schedule capabilities that help in the hardware integration and test process for the earlier incremental releases.

8. SUMMARY

It is important to recognize that the basic design of the three-component autonomy architecture—Planner/Scheduler, Smart Executive, and Fault Diagnosis and Recover—-of the Remote Agent is confirmed, The lessons presented in this paper are being factored into the current autonomy technology developments and future NMP missions. If the lessons on the organizational impact are diligently addressed, future cultural shock will be minimized. For example, on DS3, the technologists will be consultants and tool providers, while the spacecraft engineers and the mission operations team will be responsible for project-specific implementations. A much-concerted effort is put into the development of integration and testing tools. The Remote Agent is being restructured such that the three engines can be used separately and the degree of automation can be adjusted according to mission needs [4]. Finally, due to the richness of the DS1 mission, we can expect future lessons papers on the Remote Agent Experiment on DS1 and on the DS 1 mission.

ACKNOWLEDGMENTS

The work in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology under contract with the National Aeronautics and Space Administration and at the National Aeronautics and Space Administration's Ames Research Center.

The authors would like to thank James Kurien, Nicola Muscettola, Pandurang Nayak, and Brian Williams for their detail review of this paper. The authors also acknowledge the contributions of Steve Chien, Richard Doyle, Nicola Muscettola, Pandurang Nayak, Robert Rasmussen, Reid Simmons of Carnegie Mellon University, and Brian Williams for their work in leading the infusion of autonomy into the DS I flight project.

The authors would also like to acknowledge the contributions made by the members of the DS 1 autonomy/flight software team who participated in lessons learned sessions which contributed to the material in this paper.

REFERENCES

[1] D. Bernard, S. Krasner, "Integrating Autonomy Technologies into an Embedded SpacecraftSystern - Flight Software System Engineering for New Millennium," IEEE Aerospace Conference, Snowmass, CO, March 1997.

[2] B. Pen, D. E. Bernard, S. A. Chien, E. Gat, N. Muscettola, P. Nayak, M. D. Wagner, and B. C. Williams, "A Remote Agent Prototype for Spacecraft Autonomy," SPIE Proceedings Volume 2810, Denver, CO, 1996.

[3] D. E. Bernard et al, "Design of the Remote Agent Experiment for Spacecraft Autonomy," **IEEE** Aerospace Conference Proceedings, Aspen, CO, March 21-28, 1998.

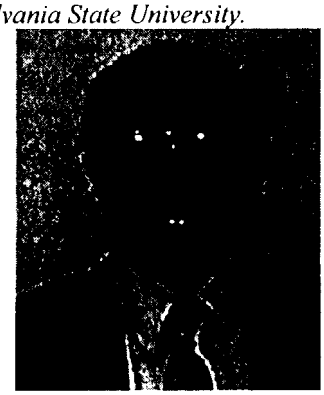
[4] B. Pen et al, "Mission Operations with an Autonomous Agent," IEEE Aerospace Conference Proceedings, Aspen, CO, March 21-28, 1998.

[5] Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects, Edward Yourdon, Prentice Hall PTR, 1997. ISBN 0-13-748310-4. Yourdon, *Death March, 1997.*

[6] M. Monternerlo, "Lessons learned from eight years of AI applications at NASA", *AI Magazine*, Winter, 1992.

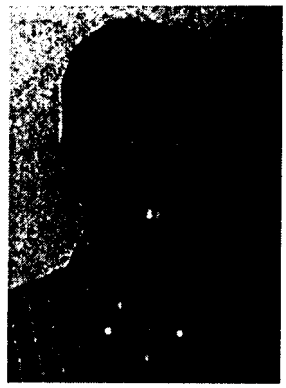
BIOGRAPHIES

***Abdullah** S. **Aljabri** received his B. Tech. in Aeronautical Engineering and Design from Loughborough University of Technology (England), and his M.S. in Aerospace Engineering from the Pennsylvania State University.*



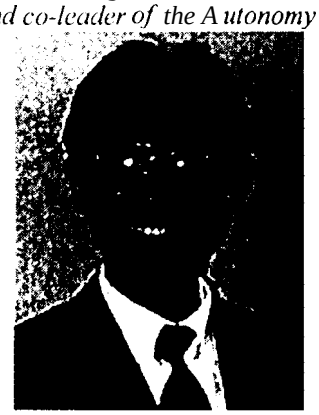
He was the Project Element Manager for the A utonomy/Flight Software development on the DS1 project. He has also served as the technical manager 0$ the Guidance and Control Research Technology Objective and Planning (R TOP) responsible for the development of advanced spacecraft control methods and avionics. Prior to joining JPL, at Lockheed Aeronautical System Company, he participated in the research and implementation of the Propfan technology and led the implementation of the P-7A flight control subsystem. His current interests include spacecraft autonomy and model based engineering.

***Dr. Douglas E. Bernard** received his B.S. in Mechanical Engineering and Mathematics from the University of Vermont, his M.S. in Mechanical Engineering from MIT and his Ph. D. in Aeronautics and Astronautics from Stanford University, He has participated in dynamics analysis and attitude control system design for several spacecraft at JPL and Hughes Aircraft, including Attitude and Articulation Control Subsystem systems engineering lead for the Cassini mission to Saturn. Currently, Dr. Bernard is group supervisor for the flight system engineering group at JPL and team lead for Remote Agent autonomy* technology development for the New Millennium Program.



***Dr. Daniel Dvorak** is a Senior Member of Technical Staff in the Information and Completing Technologies section of the Jet Propulsion Laboratory (JPL). He received a bachelor's " degree in Electrical Engineering at Rose-Hulman Institute of Technology, master's in Computer Engineering at Stanford University, and Ph. D. in Computer Science at The University of Texas at A ustin. His research at JPL has focused on monitoring, diagnosis, and system-level testing of autonomous systems, Before 1996 he was at Bell Laboratories where he worked on monitoring of telephone switching systems and on the development of R++, a rule-based extension to C++.*

***Dr. Guy K. Man** is the chairperson of the NASA New Millennium Program (NMP) Integrated Product Development Teams ([PDT) and co-leader of the A utonomy IPDT at the Jet Propulsion Laboratory, Caltech. He received his bachelor's degree in Engineering and Mathematics from the University of Redlands. He also received M.S., Engineer's Degree and Ph D. from Stanford University in Mechanical Engineering. He is currently responsible for the development and validation of break through autonomy technologies to drastically reduce mission operations cost and enable new science missions for the 21st century in the NMP. He was*



in the Guidance and Control and Avionics areas designing planetary spacecraft and developing new design and sitn ulation tools before he joined the A'All'. His interests are in autonomous systems, ystem engineering and management and in new process and tool developments. He is one of the three recipients of the 1997 NASA software of the year award.

Dr. Barney Pen is a Senior Computer Scientist in the Computational Sciences Division at NASA Ames Research Center. He is one of the architects of the Remote Agent for New Millennium 's Deep Space One (DS-1) mission, and leads a team developing the Smart Executive component of the DS- 1 Remote Agent. Dr. Pen received a B.S. degree with distinction in Symbolic Systems at Stanford University. He received a Ph. D. in computer science at Cambridge University, England, where he studied as a Marshall Scholar. His current research interests include spacecraft autonomy, integrated agent architecture, reactive execution systems, collaborative software development, and strategic reasoning. Pen was guest editor for Computational Intelligence Journal in 1996 and has given tutorials on autonomous agents, space robotics, and game e-playing.

Dr. Thomas Starbird is a Principal in System Design in the Ground Systems Section of the Jet Propulsion Laboratory (JPL). He received a B.A. from Pomona College, Claremont, California, and a Ph.D. from the University of Cal ifornia at Berkeley, both in Mathematics. He has participated in software development and in Mission Operations System development for several space projects at JPL, including the Galileo Project, where he was the Ground Software System Engineer for several years before launch. He led the development of SEQ_GEN, m ultim ission software used for constructing and checking sequences of commands to be sent to a spacecraft.